Algoritmo gerador de Expressões Regulares com base em Autômatos Finitos Determinísticos.

Fernanda Maria de Souza e Matias Giuliano Gutierrez Benitez

Centro de Ciências Tecnológicas - Universidade Estadual de Santa Catarina (UDESC)
Joinville – SC – Brasil

Abstract.

Resumo. *O seguinte trabalho apresenta um algoritmo gerador de Expressões Regulares com base em Autômatos Finitos Determinísticos (AFD). Elaborado na linguagem Java, o programa faz uso do Método Algébrico de Brzozowski, como também dos métodos de equações lineares introduzidos no denominado Lema de Arden.*

Summary. *The next document presents an algorithm that generates Regular Expressions with a base in Deterministic Finite Automaton (DFA). Elaborated in Java language, the program makes use of Brzozowski's Algebraic Method, as well as the methods of linear equations introduced in Arden's Rule.*

1. Introdução

O Algoritmo gerador de Expressões Regulares com base em Autômatos Finitos Determinístico desenvolve-se na linguagem de programação Java. Este software tem por objetivo, dado a descrição de um autômato finito determinístico (A), realizar a determinação, através da aplicação do algoritmo de Brzozowski, de uma expressão regular r tal que L(r) = L(A). Com base em informações sobre o autômato como seus símbolos (alfabeto), estados, estados finais, estados iniciais e sua tabela de transição, é gerado uma expressão regular definida a partir de conjuntos básicos e concatenação e união. Sendo um modelo algébrico, o algoritmo realiza por meio de equações lineares a conversão, e também realiza uma série de simplificações para retirar parênteses excessivos e colocar expressões em evidência para minimização de expressões regulares.

2. Contextualização

O Algoritmo gerador de Expressões regulares com base em Autômatos Finitos Determinísticos foi desenvolvido pelos graduandos Fernanda Maria de Souza e Matias Giuliano Gutierrez Benitez para o trabalho final da disciplina Linguagens Formais e Autômatos ministrada pelo Prof. Ricardo Martins do curso de graduação em Ciência da Computação da Universidade Estadual de Santa Catarina. Para desenvolvimento do algoritmo foi necessário noções lógicas e definições sobre Autômatos Finitos, Expressões Regulares, Método de Brzozowski e o Lema de Arden.

2.1. Autômatos

Um autômato é um modelo matemático de máquinas, com entradas e saídas discretas,

que reconhece um conjunto de palavras sobre um dado alfabeto. Um autômato pode ser finito ou infinito e determinístico ou não determinístico.

Um autômato finito consiste de um conjunto finito de estados, e um conjunto de transições de estados, que ocorrem a partir de símbolos de entrada escolhidos sobre um alfabeto. Se para cada símbolo de entrada existe exatamente uma transição de saída de cada estado, então o autômato é determinístico. Se existem duas ou mais transições de saída de um estado que podem ocorrer a partir do mesmo símbolo, então o autômato é definido como não determinístico.

Portanto, para desenvolvimento do software descrito anteriormente na Introdução, temos como base prioritária os Autômatos Finitos Determinísticos.

2.1.1. Autômatos Finitos Determinísticos

Os AFD's fazem parte de um modelo computacional utilizado muitas vezes em estudos teóricos-formais envolvendo Linguagens Formais e Compiladores.

Sendo um modelo que contém um número finito e pré-definido de estados, os Autômatos Finitos Determinísticos podem comumente ser definidos por uma quíntupla formada por: M= (Σ, Q, δ, q0, F). O símbolo Σ trata-se do alfabeto do autômato, que se trata de um conjunto finito de símbolos, que representam quais letras ou números foram usados para fazer as "ligações" de estados de um determinado autômato, ou seja, representados em suas arestas.

A consoante "Q" representa todos os estados presentes no autômato, ou seja, por ser um Autômato Finito Determinístico, seus estados são finitos. O símbolo δ representa a função programa, seu papel é identificar as transições possíveis em cada configuração de autômato. Uma transição s1 -0-> s2 quer dizer que se autômato está no estado s1 e o próximo símbolo da entrada é 0 então ele vai para o estado s2. Se não há mais caracteres na entrada e estamos em um estado final então o autômato aceitou a entrada. Se em algum ponto não foi possível tomar nenhuma transição, ou a entrada acabou e não estamos em um estado final, o autômato rejeitou a entrada

Por fim, 0 representa qual é o estado inicial do autômato, enquanto F representa quais são os estados finais, tendo a possibilidade de possuir mais de um.

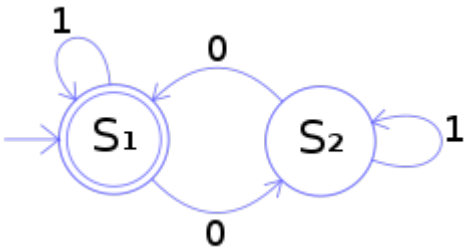


Figura 1: Autômato Finito Determinístico.

Quíntupla:

- $Q = \{S_1, S_2\}$,
- $\Sigma = \{0, 1\}$,
- $q_0 = S_1$,
- $F = \{S_1\}$,

- δ é definido por:

	0	1
S1	S2	S1
S2	S1	S2

Tabela 1: Função Programa do Autômato.

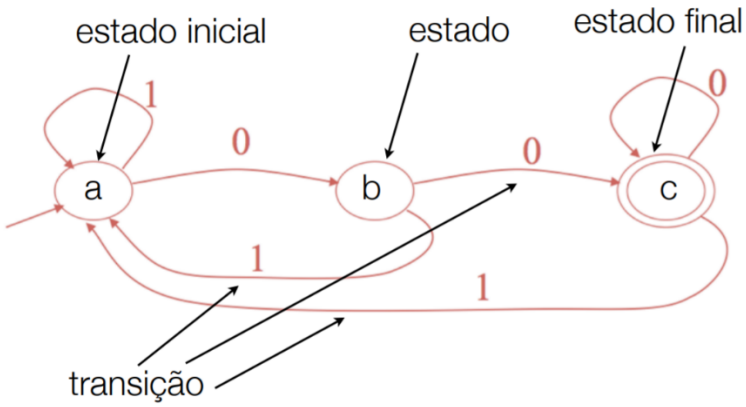


Figura 2: Exemplo de representação de estado inicial, estado contido, estado final e suas transições.

2.2. Expressões Regulares

Uma expressão regular promove uma forma concisa e flexível de identificar cadeias de caracteres, como caracteres particulares, palavras ou padrões de caracteres. O termo deriva do trabalho do matemático norte-americano Stephen Cole Kleene, que desenvolveu as expressões regulares como uma notação ao que ele chamava de álgebra de conjuntos regulares.

Ou seja, uma expressão regular (um padrão) descreve um conjunto de cadeias de caracteres, sem precisar listar todos os elementos do conjunto.

Expressões regulares são formadas basicamente, como citado na introdução, a partir de conjuntos básicos e concatenação e união.

Portanto, dado que determinado passo de indução seja: se r e s são ER e denotam as ling. R e S, então:

- União:

Denota a linguagem R ∪ S, ou seja, (r + s) é ER.

- Concatenação

Denota a linguagem R S = { uv pertence u ∈ R e v ∈ S }, ou seja, (rs) é ER.

- Concatenação Sucessiva:

Denota a linguagem R*, ou seja, (r*) é ER.

3. Descrição do Problema

O problema apresentado pelo Professor Ricardo Martins no Tema 2, caracteriza-se por elaborar uma ferramenta que possa permitir representar uma linguagem associada a um AFD, na forma de uma expressão regular. Para isso, nesse trabalho elaboramos um algoritmo, conforme informado na introdução, que utiliza o Método Algébrico de Brzozwski e o Lema de Arden para a minimização de determinado autômato e sua conversão para expressões regulares. No tópico a seguir, é explicado a ideia do algoritmo.

3.1 Método Algébrico de Brzozowski e Lema de Arden

O Método Algébrico de Brzozowski a partir de um diagrama de autômatos finitos determinísticos constrói um sistema de equações lineares. Resolvendo essas mesmas equações lineares, a solução será a expressão regular para o autômato pedido. A solução é obtida por aplicações sucessivas do Lema de Arden, ou seja, se tivermos a equação x = s + xr, então uma solução é x = sr*, para r, s ∈ a expressão regular.

Para melhores explicações, o seguinte exemplo detalha o método:

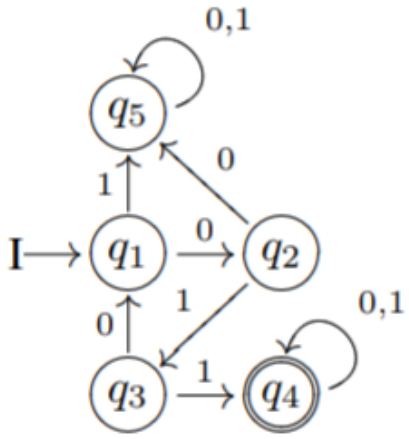


Figura 3: AFD para exemplo do Método de Brzozowski e Lema de Arden.

O alfabeto do autômato é caracterizado por {0,1}, e suas equações (tabela programa) são baseadas em:

L1 = 0L2 ∪ 1L5

L2 = 1L3 ∪ 0L5

L3 = 0L1 ∪ 1L4

L4 = 0L4 ∪ 1L4 ∪ E

L5 = 0L5 ∪ 1L5.

Uma análise no autômato mostra que q5 é um estado morto, de modo que L5 = ∅.

Com isto as equações se simplificam

$L1 = 0L2$

$L2 = 1L3$

$L3 = 0L1 \cup 1L4$

$L4 = 0L4 \cup 1L4 \cup E$

$L5 = \varnothing$.

A equação para L4 nos dá $L4 = (0 \cup 1)L4 \cup E$, de modo que precisamos aplicar o Lema de Arden. Fazendo isto obtemos:

$L4 = (0 \cup 1)* E = (0 \cup 1)*$

Substituindo em L3,

$L3 = 0L1 \cup 1(0 \cup 1)*$

De modo que, da segunda equação, segue que:

$L2 = 1(0L1 \cup 1(0 \cup 1)*) = 10L1 \cup 11(0 \cup 1)*$

Com isso, resulta da primeira equação que:

$L1 = 010L1 \cup 011(0 \cup 1)*$

Usando o Lema de Arden mais uma vez, $L1 = 010*(011(0 \cup 1)*)$, que é a linguagem aceita pelo autômato (expressão regular).

4. Modelagem do Trabalho

O trabalho é composto por 7 arquivos com diferentes funções interligadas, sendo dividido em três partes: leitura do Autômato Determinístico Finito, transformação para Expressão Regular (minimização pelo Método de Brzozowski e Lema de Arden) e por fim, a saída com a Expressão Regular equivalente a linguagem do autômato.

4.1 Especificação da Entrada

O algoritmo é todo executado pelo console, onde é feita a leitura da entrada com base na quíntupla do Autômato Finito Determinístico escolhido. O alfabeto lido pelo programa não pode conter o símbolo de vazio ou U, pois caso contrário, irar gerar problemas de compilação.

Desta forma, a entrada deve ser realizada desta maneira:

```
Digite sua AFD de acordo com as regras.

Digite os simbolos do alfabeto: 0 1
Digite os estados do automato: A B C D
Quais sao os estados finais? D
Qual o estado inicial? A
Para terminar, entre com cada transicao do AFD (ex: estadoA w estadoB):
A 1 A
A 0 B
B 1 A
B 0 C
C 1 A
C 0 D
D 1 D
D 0 D

Determinando uma expressao Regular atraves do Brzozowski

Expressao Regular: (1U1U1)*000(01)*
```

Figura 4: Exemplo de leitura de um AFD.

4.2 Melhor e Pior caraterística do código

Como melhor caraterística do código podemos definir a rapidez e simplicidade dele, e também que é capaz de fazer a transformação de qualquer autômato. E como pior caraterística podemos dizer que precisa fazer a tabela de transições para que o código funcione e que é muito sensível aos erros de digitação, por exemplo se definir uma linha que não faz sentido dentro das transições já da como resultado ERROR automaticamente.

5. Detalhes da implementação

O código em Java é elaborado principalmente com as seguintes técnicas:

- View.java

  Este arquivo armazena a função main, e esta encarregado de ler os dados da AFD e passar estes dados para o controle quem fará a distribuição dos mesmos. Além disso, também imprime o resultado da expressão regular.

```
15
16      public static void main(String[] args) {
17
18          System.out.println("Digite sua AFD de acordo com as regras.\n");
19          lerConsole();
20          System.out.println("Determinando uma expressao Regular atraves do Brzozowski");
21          System.out.println("\nExpressao Regular: "+control.brzozowski()+"\n");
22      }
23
```

Figura 5: Função view.main

```java
private static void lerConsole() {
    control.inicializaAutomato();

    int i,total;
    String s;
    StringTokenizer st;

    System.out.print("Digite os simbolos do alfabeto: ");
    st = new StringTokenizer(readString());
    total = st.countTokens();
    for (i=0; i<total; i++) control.addSimbolo(st.nextToken());

    System.out.print("Digite os estados do automato: ");
    st = new StringTokenizer(readString());
    total = st.countTokens();
    for (i=0; i<total; i++)
        control.addEstado(st.nextToken());

    System.out.print("Quais sao os estados finais? ");
    st = new StringTokenizer(readString());
    total = st.countTokens();
    for (i=0; i<total; i++)
        control.setEstadoFinal(st.nextToken());

    System.out.print("Qual o estado inicial? ");
    control.setEstadoInicial(readString());

    System.out.println("Para terminar, entre com cada transicao do AFD (ex: estadoA w estadoB):");
    for (i = 0; i < control.numeroTransicoes(); i++) {
        s = readString();
        st = new StringTokenizer(s);
        control.addTransicao(st.nextToken(),st.nextToken(),st.nextToken());
    }
    System.out.println("");
    menu();
}

```

Figura 6: Função para ler o console.

- Controle.java

O arquivo View.java vai lendo e passando os dados necessários do AFD para este arquivo, então o objetivo de cada função do Controle.java é enviar cada dado para o lugar certo, como diz seu nome é simplesmente um arquivo de controle de envio de dados e uma questão de órganizacacão.

```java
public class Controle {

    private AutomatoFinitoDeterministico afd;

    public Controle() {}

    public void inicializaAutomato() {
        afd = new AutomatoFinitoDeterministico();
    }

    public void addSimbolo(String s) {
        afd.addSimbolo(s);
    }

    public void addEstado(String e) {
        afd.addEstado(e);
    }

    public void setEstadoFinal(String e) {
        afd.setEstadoFinal(e);
    }

    public void setEstadoInicial(String e) {
        afd.setEstadoInicial(e);
    }

    public void addTransicao(String ea, String s, String ep) {
        afd.addTransicao(ea,s,ep);
    }


    public int numeroTransicoes() {
        return afd.getEstados().size()*afd.getAlfabeto().size();
    }

    public String brzozowski() {
        Brzozowski br = new Brzozowski(afd);
        return br.brzozowski();
    }
}
```

Figura 7: Classe Controle

- AutômatoFinitoDeterminístico.java

Para a alocação das informações lidas pelo console, é criado uma grande classe chamada "AutomatoFinitoDeterministico" contendo o alfabeto, estados, estados finais, estado inicial e estado atual, sendo estes três primeiros guardados em uma estrutura de dados de tipo TreeSet (Árvore) para facilitar ser encontradas e implementados nas transformações.

```
private TreeSet<Simbolo> alfabeto;
private TreeSet<Estado> estados;
private TreeSet<Estado> estadosFinais;
private Estado estadoInicial;
private Estado estadoAtual;
```

São criadas funções que fazem a definição dos estados iniciais, novos estados, novas transições, e novos estados finais a cada remanejo que o algoritmo implica, como também novos símbolos que serão agregados no alfabeto.

```
19  public void setEstadoInicial(String estado) {
20      for (Estado e: estados) {
21          if (e.getEstado().compareTo(estado) == 0) estadoInicial = e;
22      }
23      estadoAtual = estadoInicial;
24  }
25
26  public void addEstado(String s) {
27      Estado novo = new Estado(s);
28      estados.add(novo);
29  }
30
31  public void addTransicao(String estado1, String simbolo, String estado2) {
32      Estado ei = null;
33      Estado ef = null;
34      Simbolo s = null;
35      for (Estado e: estados) {
36          //Pega os estados
37          if (e.getEstado().compareTo(estado1) == 0) ei = e;
38          if (e.getEstado().compareTo(estado2) == 0) ef = e;
39      }
40      for (Simbolo sim: alfabeto) {
41          //Pega o simbolo
42          if (sim.getSimbolo().compareTo(simbolo) == 0) s = sim;
43      }
44      //Adiciona a transicao
45      if (ei != null && ef != null && s != null) ei.addTransicao(s, ef);
46  }
47
48  public void setEstadoFinal(String s) {
49      for (Estado e: estados) {
50          if (e.getEstado().compareTo(s) == 0) {
51              e.setFinal();
52              estadosFinais.add(e);
53          }
54      }
55  }
```

- Brzozowski.java

Neste arquivo é feita a inicialização do AFD, colocando nele os estados com suas respectivas linguagens, para posterior execução.

```java
public class Brzozowski {
    private AutomatoFinitoDeterministico afd;
    private TreeMap<Integer, Linguagem> linguagens;
    private int qtd;

    public Brzozowski(AutomatoFinitoDeterministico afd) {
        this.afd = afd;
        linguagens = new TreeMap<Integer, Linguagem>();
        int i = 1;
        for (Estado e: this.afd.getEstados()) {
            if (e.compareTo(afd.getEstadoInicial()) == 0) {
                linguagens.put(1,new Linguagem(e.getLinguagem(), e.getEstado()));
            } else {
                ++i;
                linguagens.put(i, new Linguagem(e.getLinguagem(), e.getEstado()));
            }
        }
        qtd = i;
    }
```

Figura 8: Implementação dos estados do Autômato Finito Determinístico com sua linguagem.

```java
    public String brzozowski() {
        for (int i = qtd; i > 0; i--) {

            if (i == qtd) {
                if (!linguagens.get(i).verifica()) {
                    linguagens.get(i).setLinguagem(linguagens.get(i).simplifica(linguagens.get(i).getLinguagem()));
                    linguagens.get(i).arden();
                }
            } else {
                for (int j = qtd; j > i; j--) {
                    linguagens.get(i).substitui(linguagens.get(j).getEstado(),linguagens.get(j).getLinguagem());
                }
                if (!linguagens.get(i).verifica()) {
                    linguagens.get(i).setLinguagem(linguagens.get(i).simplifica(linguagens.get(i).getLinguagem()));
                    linguagens.get(i).arden();
                }
            }
        }

        return linguagens.get(1).getLinguagem();
    }
}
```

Figura 9: Execução do Método de Brzozowski.

- Linguagem.java

Este arquivo contém a função do teorema de Arden, que é aplicado no Método de Brzozowski.

```java
    public void arden() {
        String A = "", B = "";
        String[] temp, splitL;
        String[] split = linguagem.split("\\(|\\)");

        for (int i = 0; i < split.length; i++) {
            if (split[i].contains("L")) {
                if (split[i].compareTo("L") == 0 && split[i+1].compareTo(estado) == 0) {
                    A = split[i-1];
                    break;
                }
                if (split[i+1].compareTo(estado) == 0) {
                    temp = split[i].split("U");
                    A = temp[temp.length-1].replace("L", "");
                    break;
                }
            }
        }
        B = linguagem.replace(parentesis(A)+"L("+estado+")","");
        B = B.replace("UU", "U");

        splitL = B.split("U");
        B = "";
        for (String s: splitL) {
            if (!s.equals(")") && !s.equals("(")) {
                if (!B.equals("")) B += s;
                else B = s;
            }
        }

        A = parentesis(A);
        A += "*";

        B = parentesis(B);

        if (!B.equals("({E})")) linguagem = A + B;
        else linguagem = A;
```

Figura 10: Método de Arden

Neste arquivo, também é implementada uma função de simplificação que realiza a retirada de parênteses desnecessários.

```
110     private String parentesis(String s) {
111         boolean temParentesis = false;
112         if (!s.equals("")) temParentesis = s.charAt(0) == '(' && s.charAt(s.length()-1) == ')';
113         if (!temParentesis) {
114             if (s.length() > 1 && !s.contains("L(")) {
115                 s = "(" + s + ")";
116             }
117         } else {
118             s = s.substring(1, s.length()-1);
119         }
120         return s;
121     }
122
```

Figura 11: Retirada de parênteses excessivos.

E também, outra simplificação é a exibição em evidência para imprimir uma expressão regular.

```
22      public String simplifica(String ling) {
23          String[] splitU = ling.split("U");
24          int cont = 0;
25          for (String s: splitU) {
26              if (s.contains("L(" + estado + ")")) cont++;
27          }
28          if (cont <= 1) {
29              isSimplificado = true;
30          } else {
31              String temp = "";
32              String A = "";
33              for (String s: splitU) {
34                  if (s.contains("L(" + estado + ")")) {
35                      if (A == "") A = s.replace("L(" + estado + ")", "");
36                      else A += "U" + s.replace("L(" + estado + ")", "");
37                  }
38              }
39              A = parentesis(A);
40              A += "L(" + estado + ")";
41              for (int i = 0; i < splitU.length; i++) {
42                  if (!splitU[i].contains("L(" + estado + ")")) {
43                      if (temp.equals("")) temp = temp.concat(splitU[i]);
44                      else temp = temp.concat("U" + splitU[i]);
45                  }
46              }
47
48              if (temp.contains("U") || (temp.length() != 1 && !temp.equals("") && !temp.equals("{E}"))) temp = "(" + temp + ")";
49              if (temp.equals("")) {
50                  ling = A;
51              } else {
52                  ling = A + "U" + temp;
53              }
54              isSimplificado = true;
55          }
56          return ling;
```

Figura 12: Simplificação da expressão regular.

- Simbolos.java

A classe símbolos esta encarregada de todas as funções que utilizam o símbolo do alfabeto principalmente como comparação. Para desta maneira facilitar o acesso aos Símbolos do alfabeto.

```
7
8       private String simbolo;
9
10      public Simbolo(String simbolo) {
11          this.simbolo = simbolo;
12      }
13
14      public String getSimbolo() {
15          return simbolo;
16      }
17
18      public void setSimbolo(String simbolo) {
19          this.simbolo = simbolo;
20      }
21
22      @Override
23      public int compareTo(Simbolo o) {
24          return simbolo.compareTo(o.getSimbolo());
25      }
26
27      @Override
28      public String toString() {
29          return simbolo;
30      }
31  }
```

Figura 13: todas as funções de comparação da classe símbolo.

- Estado.java

A cada interação do autômato para minimização do mesmo, é retornada a função para definir de acordo com o novo estado inicial e o novo estado final, a nova linguagem a ser calculada pelo Método.

```
19      public String getLinguagem () {
20          String L = "";
21          int i = 0;
22          Set<Simbolo> keyset = transicoes.keySet();
23          Iterator<Simbolo> it = keyset.iterator();
24          while (it.hasNext()) {
25              Simbolo s = it.next();
26              if (it.hasNext()) {
27                  L = L.concat(s.toString() + "L(" + transicoes.get(
28                  i += transicoes.get(s).toString().length() + 2;
29              } else {
30                  L = L.concat(s.toString() + "L(" + transicoes.get(
31                  i += transicoes.get(s).toString().length() + 1;
32              }
33          }
34          if (isFinal) L = L.concat("U{E}");
35          return L;
36      }
37
```

Figura 14: Função para retornar a linguagem.

O símbolo da transição também é definido pela função de acordo com os estados subsequentes.

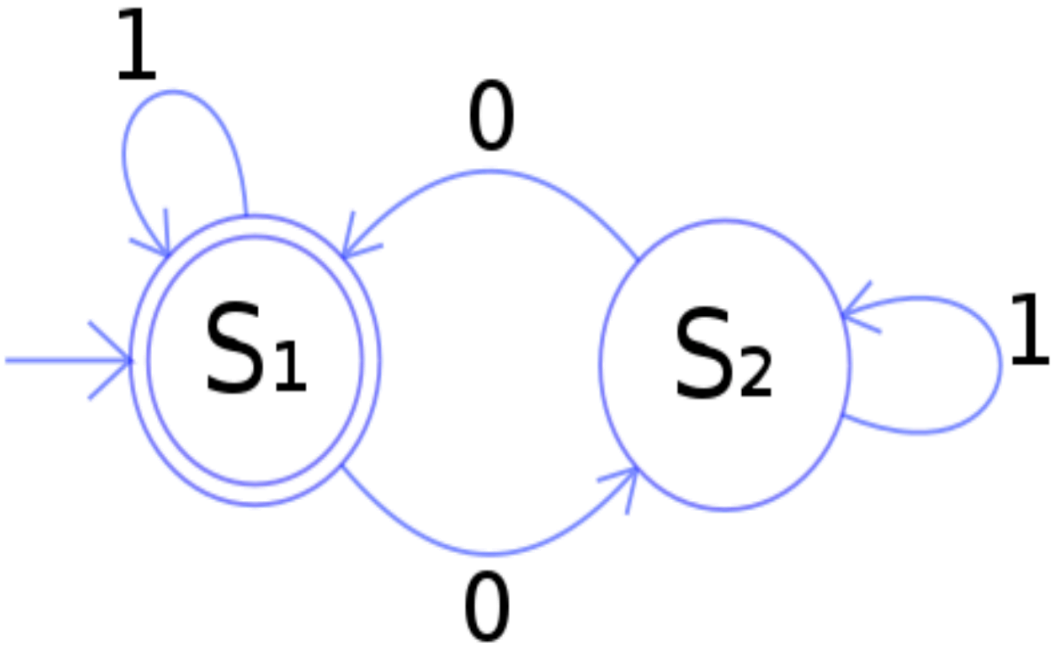
```
40      public String simboloTransicaoPara(Estado e) {
41          String st = "";
42          if (transicoes.containsValue(e)) {
43              for (Simbolo s: transicoes.keySet()) {
44                  if (transicoes.get(s).getEstado().equals(e.getEstado())) {
45                      if (st.equals("")) {
46                          st = s.getSimbolo();
47                      } else {
48                          st += "U" + s.getSimbolo();
49                      }
50                  }
51              }
52          }
53          if (st.equals("")) return "{}";
54          return st;
55      }
56
```

Figura 15: Função para retornar o símbolo da transição.

6. Testes

Testes realizados para confirmação do sucesso do algoritmo.

- Teste 1:



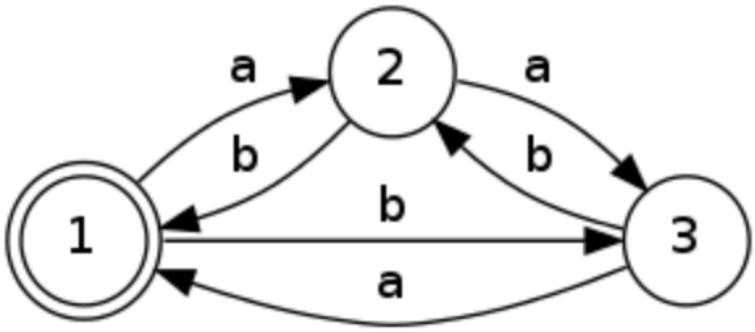

```
Digite sua AFD de acordo com as regras.

Digite os simbolos do alfabeto: 0 1
Digite os estados do automato: a b
Quais sao os estados finais? a
Qual o estado inicial? a
Para terminar, entre com cada transicao do AFD (ex: estadoA w estadoB):
a 0 b
a 1 a
b 0 a
b 1 b

Determinando uma expressao Regular atraves do Brzozowski

Expressao Regular: (01*0U1)*
```

- Teste 2:




```
Digite sua AFD de acordo com as regras.

Digite os simbolos do alfabeto: a b
Digite os estados do automato: 1 2 3
Quais sao os estados finais? 1
Qual o estado inicial? 1
Para terminar, entre com cada transicao do AFD (ex: estadoA w estadoB):
1 a 2
1 b 3
2 a 3
2 b 1
3 a 1
3 b 2

Determinando uma expressao Regular atraves do Brzozowski

Expressao Regular: (ab*aabUbaUbb*aab)*
```

7. Conclusão

Por fim, com o desenvolvimento do algoritmo proposto e o estudo de métodos para minimização, transformação, leitura, representação e exposição de autômatos e expressões regulares, chegamos à conclusão que o projeto não se limita somente a teoria, podendo ser aplicado no meio prático em situações reais, como linhas de produção. Outro ponto que destaca-se possui relação direta com os métodos matemáticos utilizados, sendo estes o de Brzozowski e o Lema de Arden, que serviram de base para o trabalho e ponto fixo para todas as outras ideias surgirem e formarem um programa funcional que a partir de uma quíntupla de qualquer autômato finito determinístico, gera uma expressão regular equivalente.

Ainda sobre a visão lógica, resolvemos utilizar a linguagem de programação Java para uma implementação mais simples do algoritmo, tendo por critério a criação de árvores: usando a linguagem C (a qual foi nossa primeira opção) a projeção de árvores é criteriosamente mais complexa, visto que em Java apenas com uma declaração podemos realizar a criação da mesma, além disto a utilização de classes facilito o desenvolvimento do código, e também a aplicação dos algoritmos.

Portanto, após semanas de pesquisa saímos com conhecimentos abrangentes sobre

autômatos e expressões regulares, bem como métodos para representação em algoritmos dos mesmo, visando a facilitação de seu uso em diversas áreas do conhecimento.

8. Referências

HOW to convert finite automata to regular expressions?. [S. l.], 2012. Disponível em: https://cs.stackexchange.com/questions/2016/how-to-convert-finite-automata-to-regular-expressions. Acesso em: 31 maio 2019.

LEVET, Michael. **Brzozowski Algebraic Method**. [S. l.], 2015. Disponível em: https://www.dreamincode.net/forums/index.php?app=core&module=attach§ion=attach&attach_id=38797. Acesso em: 29 maio 2019.

AUTÓMATOS Finitos. [S. l.], 2014. Disponível em: http://www.dcc.fc.up.pt/~rvr/resources/MC/C4.pdf. Acesso em: 29 maio 2019.

AFNS, Operações Regulares e Expressões Regulares. [S. l.], 2019. Disponível em: http://www.decom.ufop.br/anderson/BCC242/REX.pdf. Acesso em: 29 maio 2019.

COUTINHO, S.C. **Autômatos e Linguagens Formais**. Rio de Janeiro, 2007. Disponível em: https://www.dcc.ufrj.br/~collier/e-books/LF.pdf. Acesso em: 29 maio 2019.

DE MELO, Raul Felipe. **Entendendo de uma vez por todas Expressões Regulares: Parte 1 — Introdução**. [S. l.], 2017. Disponível em: https://medium.com/trainingcenter/entendendo-de-uma-vez-por-todas-expressões-regulares-parte-1-introdução-dfe63e289dc3. Acesso em: 30 maio 2019.

PALAZZO, Luiz A M. **Linguagens Regulares e Autômatos Finitos**. Pelotas, 2011. Disponível em: http://infocat.ucpel.tche.br/disc/lfa/docs/LFA-T01.pdf. Acesso em: 1 jun. 2019.

DOMINGUES PRADO, Simone das Graças. **Linguagens Regulares e Autômatos Finitos**. [S. l.], 2008. Disponível em: http://wwwp.fc.unesp.br/~simonedp/zipados/TC02.pdf. Acesso em: 1 jun. 2019.

PATI, Aditya. **Arden's Theorem and Challenging Applications | Set 2**. [S. l.], 20015. Disponível em: https://www.geeksforgeeks.org/ardens-theorem-and-challenging-applications-set-2/. Acesso em: 1 jun. 2019.